Coursework C

Introduction

The analysis of electrode recordings of neurons is an interesting area of research in neurology. When neurons fire, an electrode nearby can pick up on the electric signal produced and record it into a digital signal where it is recorded as a spike in the signal. These electrodes in the brain capture the signal surrounding them, so can capture the spikes from multiple neurons around it.

Analysis of the recording allows researchers to understand the activity of neurons. Given that the digital recording can contain the signal from multiple neurons as well as background noise from the brain and environment, it is important for researchers to somehow filter the recording and analyze it to find the spikes produced by different neurons which can potentially overlap. This process is known in neuroscience as Spike Sorting. This project works on data provided by a single electrode and as such cannot use modern techniques such as comparing the signal from two or more nearby electrodes to extract. Consequently, more analytical techniques are implemented to filter and detect the spikes in the recording. As mentioned in the Scholarpedia article regarding Spike sorting (Quiroga, 2007), there are 4 steps to spike sorting: filtering, spike detection, feature extraction and clustering which were implemented for this project.

Signal Filtering & Spike Detection

The first step to spike sorting is to filter the digital recording to remove any unwanted noise and low-frequency variations in the dataset from obstructing the following steps from performing correctly and accurately. In the case of this project, we have 2 digital recordings: the recording for training and the recording for submission.

Plotting small samples from both recordings (training top, submission bottom) in Figure 1, it is clear that they include a lot of high-frequency noise which can be a problem for spike detection as it can hide actual spikes and require a higher amplitude for detecting spikes, potentially reducing the number found to ensure accuracy in the spikes detected.

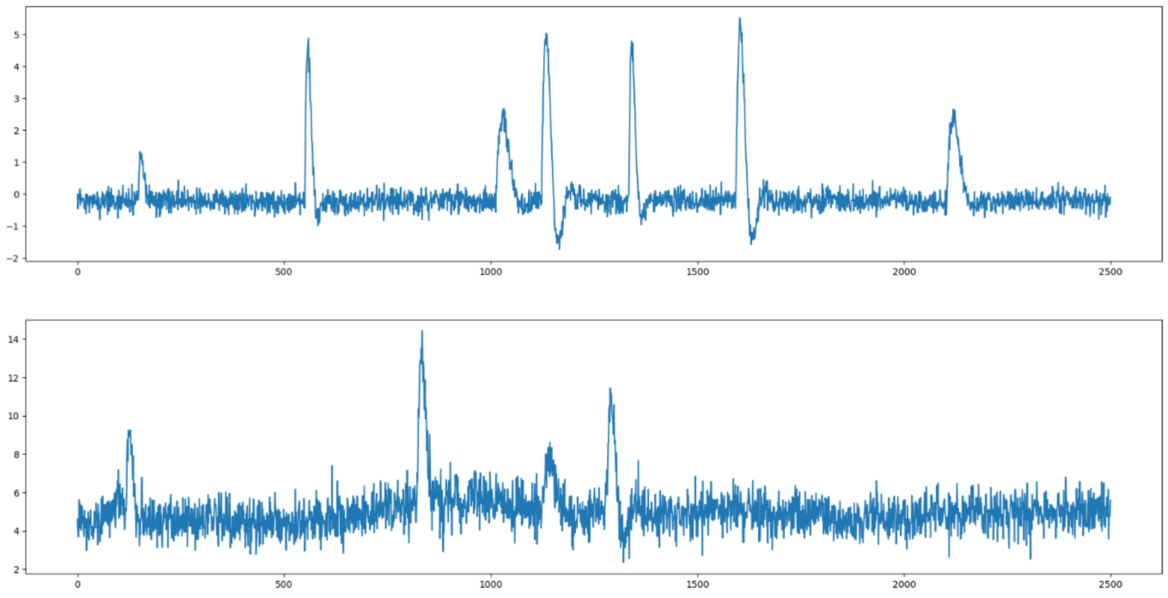

Figure 1: 2500 samples from training (top) and submission (bottom) recordings

To combat this a low pass filter was required to attenuate the high frequencies and allow the real spikes to be easily visualized and detected by the following step. Through testing, a frequency boundary of 2500Hz provided the best filtering whilst retaining the general shape of spikes, even the smallest ones. The result of the low pass filter can be seen in Figure 2 below.

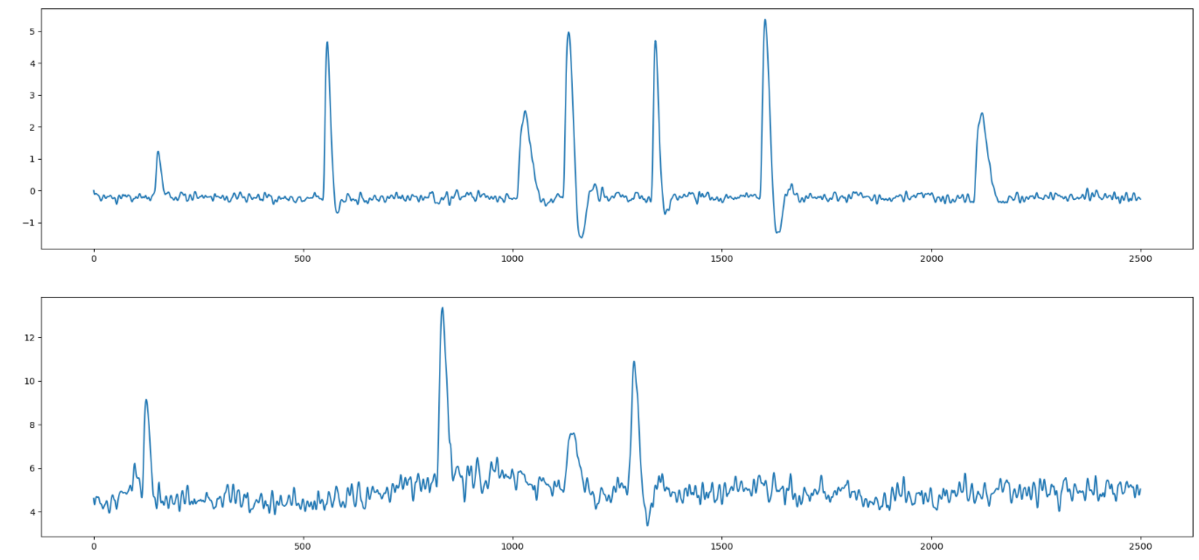

Figure 2: Samples from Figure 1 with low pass filter

The submission recording had more noise than the training recording, so a lower frequency boundary was required at 1800Hz for the submission recording to better visualize spikes. Additionally, when plotting the entire recordings, it is clear that the submission recording also has low-frequency variation likely caused by the subject of the recordings moving during the recording. This can be seen in the difference between the training (top) and submission (bottom) recording in Figure 3 below.

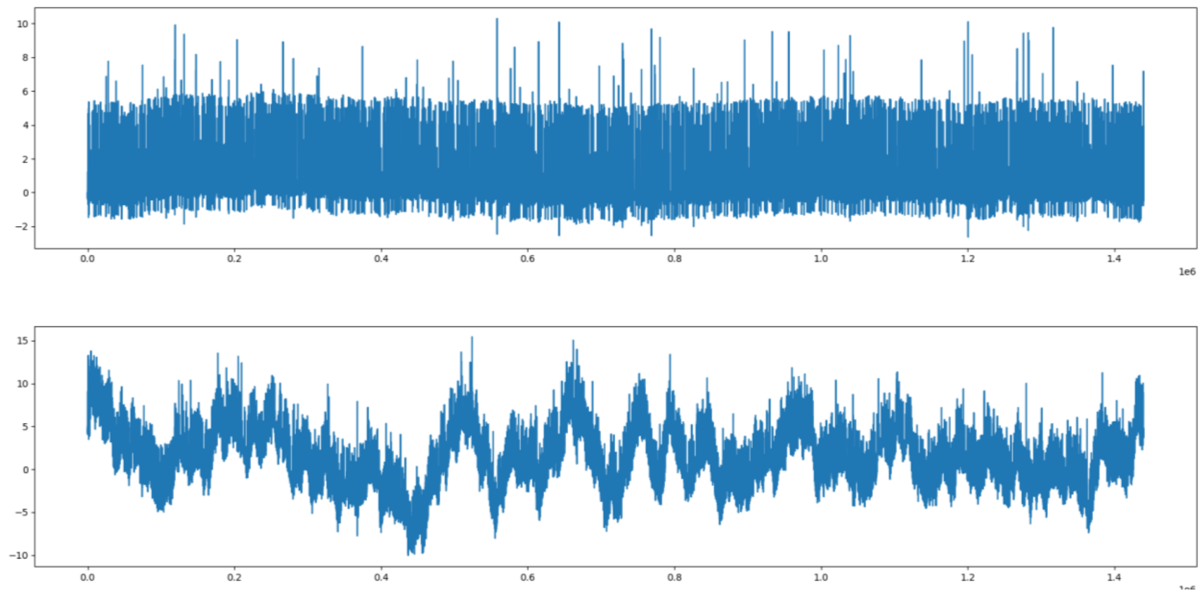

Figure 3: Entire electrode recordings, training (top) & submission (bottom)

To remove this low-frequency noise in the submission recording, a high pass filter is also required for it where a low frequency of 100Hz works well for normalizing the recording to the same shape as the training recording. Consequently, the two filters for the submission recording were replaced by a band-pass filter with the original high pass and low pass boundary frequencies.

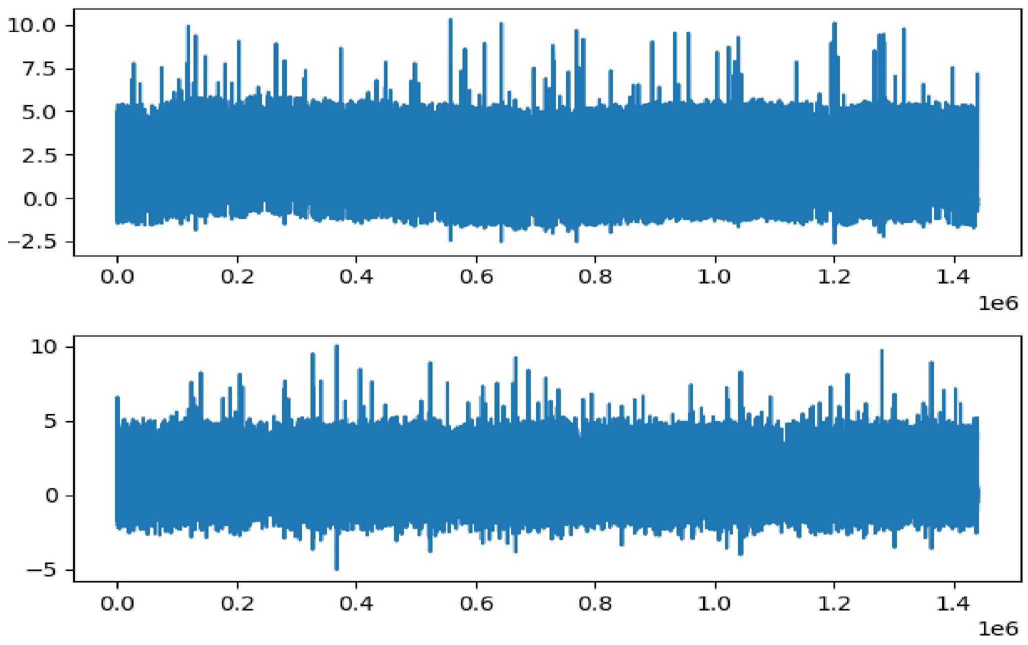

Figure 4: Same entire recordings from Figure 3. Submission passed through high pass filter

Spike Detection

Once the recordings have been filtered, the next step is to detect the spikes in the data.

Rey et al.'s 2015 paper and Mokri et al. (Rey, Pedreira and Quian Quiroga, 2015; Mokri *et al.*, 2017) recommends using the absolute median deviation multiplied by a constant between 3 and 5 is a very robust method for peak detection with spikes having little effect on it. The result of this formula is the height threshold to measure peaks.

$$c * \hat{\sigma} = c * \frac{median(|X|)}{0.6745}$$

In this way, as left-over noise will be the majority of samples and have the most influence on standard deviation, spikes should be outside the range of this deviation and therefore be detected.

Through testing, it became apparent that even with the best efforts to avoid false positive detections (detecting peaks that are not neuron spikes), some occur using this method to the best of my knowledge. Increasing the constant meant that these false positives are greatly reduced but that some smaller spikes are missed. The opposite is true if the constant is smaller, increasing the spikes detected but also the false positives. Through testing, 3.7 provided the best balance between TPs and FPs. In the training recording, it finds 3242 peaks out of 3343 in the training set. 3225 are of real spikes so a precision of 99.5% and sensitivity of 96.5%.[1]

[1] Terms precision and sensitivity are explained in the Performance Metric section

Constant	3	3.4	3.7	4	4.5
TP	3233	3230	3225	3213	3169
Sensitivity	96.71%	96.62%	96.47%	96.11%	94.8%
FP	93	23	17	12	9
Precision	97.20%	99.29%	99.48%	99.63%	99.72%

With the training recording, since the classes and indexes of real spikes was known, it was possible to remove duplicate or erroneous spike detections to only pass onto the classification algorithms, real spikes. To pass on the data for each spike to the following steps, I decided that I would pass on the amplitude reading of a window of samples around the spike's peak. Having detected the peak of the spikes, the width of the spikes could then be calculated. It was calculated to 0.9 prominence to minimize outliers and errors from spikes that have very big dips and would have incredibly large widths at full prominence. The median width, to avoid any remaining outliers, was around 50 samples a good starting point to analyze the spikes. I plotted the average spike for each neuron class with 45 samples to the left and right of the peak to get a better picture of them than with a width of 50. Only 15 samples are needed to the left of the peak to draw the spike with the right requiring more with 35 samples allowing an overview of most of the spikes, especially wider class 4 and 5.

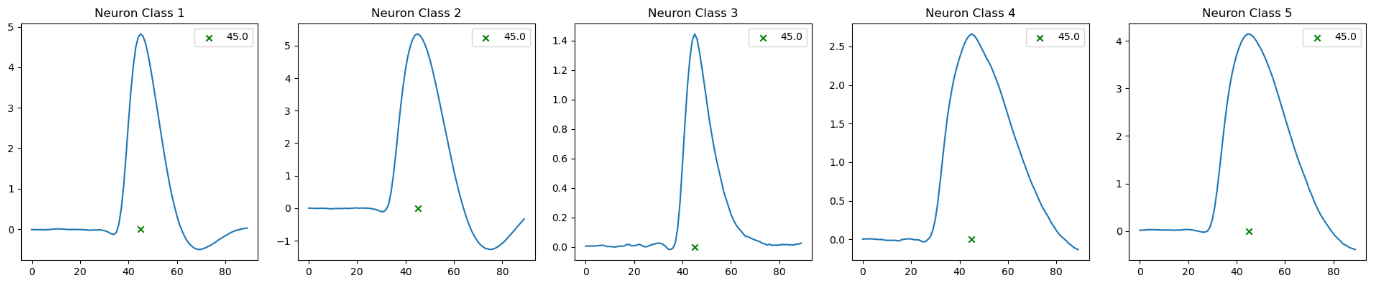

Figure 5: Average spike for each neuron class

Feature extraction

The next step is feature extraction. Whilst it is possible to pass on the raw data of each peak to the classification algorithms, it is highly recommended to use feature extraction to reduce the number of variables and dimensions per spike due to large processing and storage requirements otherwise.

Principal Component Analysis is a very popular and widely utilized method to extract features that is easy to implement in python and as such was chosen for this task based on Rey, Pedreira and Quiroga's paper (Rey, Pedreira and Quian Quiroga, 2015) which highlights its popularity and success. Using PCA, 99% of the variance of the spikes was extracted using just 6 dimensions instead of the original 50. This greatly reduced the complexity of the algorithms and the time required to train them.

Below in Figure 6, the training data was transformed with just the 2 principal components representing the largest variance and plotted to visualize its effectiveness. There are clear clusters that can be detected for each class of neuron, and this gives confidence that using PCA along with the classification techniques will provide greater performance.

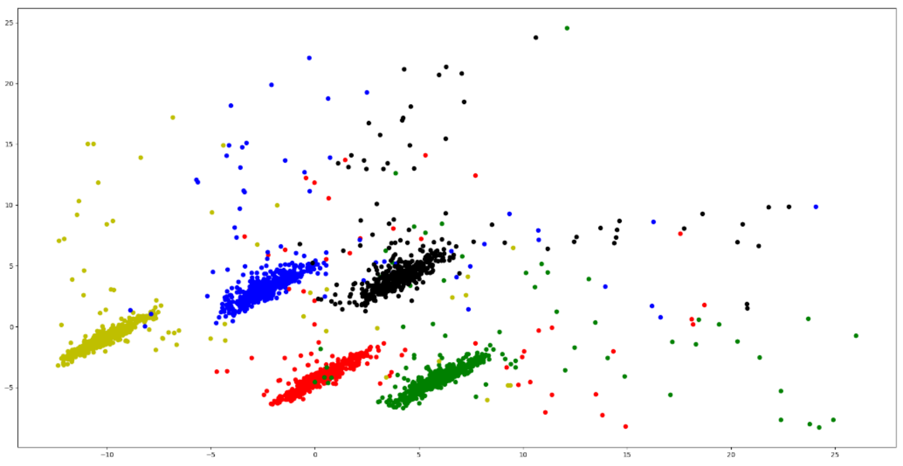

Figure 6: Plot of 2 PCA components of training data, coloured by neuron class

Classification/Clustering

Performance metrics

Before implementing the algorithms and analyzing their performance it is important to understand and outline how that performance will be measured. In other words, what will the performance metrics be?

Researching published papers and popular data science blogs reveals that there are various common metrics for CI classification algorithms. These metrics are accuracy, precision, sensitivity or recall and F1 score (Sunasra, 2019). To understand these, it is important to understand a confusion matrix. This shows the class predictions by a classification algorithm divided into the groups True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). It is a bit more complex due to having 5 neuron classes, but TP represents the number spikes correctly classified as their correct neuron class. TN is the number of spikes correctly classified as other neuron classes. FP is the spikes from other neuron classes incorrectly classified as a particular one. And FN would be spikes from a particular class incorrectly classified as another.

Knowing this, accuracy is the percentage of correct classification predictions by a model. Precision is the proportion predictions for a certain class that are correct. Sensitivity is the proportion of spikes of a certain class that are correctly identified.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad Precision = \frac{TP}{TP+FP} \qquad Sensitivity = \frac{TP}{TP+FN}$$

From precision and sensitivity, the F1 score is calculated to simplify these metrics and have a general score. It represents the harmonic mean of the two metrics meaning that it is close to both when they are almost equal and close to the smaller metric when they are different (Shmueli, 2020). While F1 score can be considered a bad measure of performance when recall and precision do not carry the same importance. This is not the case currently, as False Positives and False Negatives for spike classification represent the same outcome, a mislabeled spike. As such, the F1 score presents a clearer view of performance where both scores are important, not just focusing on precision by minimizing False Positives or recall by reducing False Negatives.

$$F1 = 2 * \frac{Precision * Sensitivity}{Precision + Sensitivity}$$

Due to having multiclass classification, the F1 score needs to be calculated based on the F1 score for each class. Given that the proportional distribution of spikes for each class, the macro F1 score was used which is the mean of the 5 classes' F1 score.

$$Macro\ F1 = \frac{F1_1 + F1_2 + F1_3 + F1_4 + F1_5}{5}$$

To find these performance metrics it is necessary to test the models with testing data. The only labelled data for this task is the training data and as such will need to be split into training and testing data to avoid any bias and maximize the accuracy of the metric's result. A common split is 80-20, 80% for training and 20% for testing. The use of Scipy's train_test_split() function to perform the split meant that each run of the model's had random train and test samples, maintaining the same split percentage, to avoid any bias from overfitting for specific test data.

First Solution: Artificial Neural Network

Artificial neural networks (NN) are a CI technique that utilizes neurons or nodes represent the action of a function, the result of which can be considered a prediction or classification for example. NNs can be trained with the weights of neurons being changed so that the output from an input matches a target. Neural nets can approximate any given function and as such can provide a great way to predict classifications. From feature extraction, the neuron classes form clusters that can be easily identified. Therefore, training the neural network should allow it to regress to a function that can differentiate and classify each neuron class.

Throughout testing, it became apparent that the neural network performance with only the 6 PCA inputs had a lot more variation and lower performance based on the chosen performance metric. Given all of this and the fact that neural networks are great tools for finding patterns in data with many dimensions and many samples, both of which are true for this task, using a neural network is a great option to test spike classification with the full peak window as input.

For a neural network, the shape of the neural network, for example, the number of layers, is a very important consideration. Given the fact that it will have 50 inputs, it was decided to go with 1 hidden layer with 45 nodes for testing purposes. Additionally, the learning rate was fixed at 0.001.

For classification with Neural Nets, it is common to have the activation function for the output layer to be the Softmax function, which transforms the outputs of the output nodes into probabilities to easily understand and utilize the likelihood/confidence of classifications from the neural net.

Training the neural net on the training data resulted in the performance on the testing data below:

Test #	1	2	3	4	5
F1 Score	95.53	95.38	91.65	94.66	95.25

It shows quite positive performance with 94.5% on average and is a good candidate for further optimization with the number of hidden nodes and learning rate being possible tunable hyperparameters.

Second Solution: K-Nearest Neighbours

KNN is another popular classification algorithm. The best way to visualize it is that it plots the training data points in an N-dimension space (N being the number of features of each data point). Each data point has a class associated with it. When a new prediction is to be made, the new data point is inserted into that data space and the distance to all training points is calculated. This distance function can vary based on the p norm. It represents the power to which the difference between the points is elevated. For example, for two points x = [x1,x2…,xn] and y = [y1,y2,….,yn].

$$d = \sqrt[p]{|x_1 - y_1|^p + |x_2 - y_2|^p + \cdots + |x_n - y_n|^p}$$

The K in KNN stands for the number of points nearest to the new data point that are considered to decide the class of the new point. The class with the most points is the predicted class of the new data point. This method is great for classification as there is no need to understand or calculate the underlying relationships, simply matching a new sample to the training samples most like it. From feature extraction and Figure 6, it is already clear that the neuron classes form clusters, and this is where KNN succeeds and as such this classification algorithm is a great fit for this problem. Another very popular classification algorithm Support Vector Machine (SVM) was tested but had similar performance to KNN and less optimizable parameters, therefore KNN was implemented and tested. K=10 and p=2 were set as the default parameters.

Using this method with the training recording resulted in the following performance.

Test #	1	2	3	4	5
F1 Score	97.69	96.58	97.08	96.78	96.61

This is a great performance by the KNN, with an average of 97% and shows that there are very few cases where it struggles with classification.

Optimization

Given the performance of the two classification algorithms, KNN has superior performance and was faster to set up and train than the neural network in practice. For that reason, it was chosen to optimize the K-Nearest Neighbours algorithm and, as mentioned previously, KNN has two variables that can be optimized, K and p.

Simulated Annealing (SA) was chosen for the optimization algorithm due to how it quickly regresses to the global maximum and little memory requirements. In the blog Glassbox, Computer Science PhD Rachael Draelos (Draelos, 2019) highlights how optimizing an algorithm based on the score from testing or evaluation data leads to overfitting. Consequently, another split in the training subset should be used, splitting into a training and validation subset for SA.

Utilizing SA resulted in the optimal values of K = 1 and p around 1-3. Using one nearest neighbour works well with the training data but the submission recording is different and includes slightly more noise and variation. This means there's a chance K = 1 could be overfitting to the training data. To solve this, the number of nearest neighbours was increased to 3, to reduce the probability of error from overfitting and have greater confidence with classifications. At least 2/3 of the closest neighbours will be of that class, not just a potential outlier. This reduces the performance using test data slightly but as mentioned, will be useful when classifying unknown spikes. Additionally, the variation in p depending on the training-validation split does not affect final performance on test data. Using these parameters for the complete training data set and testing on the testing split achieved the performance below:

Test #	1	2	3	4	5
F1 Score	97.98	97.65	98.09	97.96	97.49

Performance has increased to 97.8% on average from the simple KNN as well as less variation in performance from test to test. Therefore, using SA has optimized the KNN and will likely result in greater performance in classifying the submission recording.

Conclusion

Spike sorting is an important area of research in neuroscience with a wide range of applications that require neuron scans such as brain scans, memory studies and many others and research is ongoing to improve the technique. The spike sorting process was performed with 2 different classification algorithms, a Neural Network and KNN to spike sort the recording of a single electrode in a mouse's brain. The result of using these two algorithms was 94.5% and 97% on average respectively. KNN was further optimized using simulated annealing to optimize the parameters and its performance to an average of 97.8% with the test data. This optimum algorithm was then used to perform spike sorting on the filtered and peaks detected in the submission recording.

Plotting the average neuron spike from the training recording and comparing it with the class predictions in Figure 7, shows a very close shape and structure. Some outliers are less certain but overall the predictions look quite promising and I am personally confident that its classification performance is relatively high.

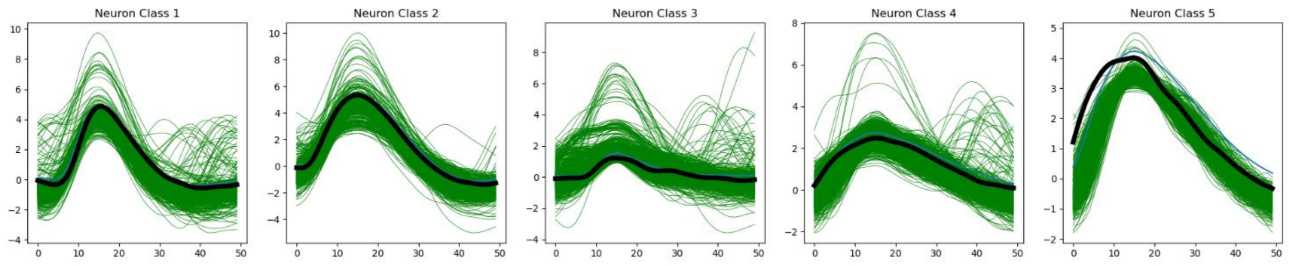

Figure 7: Plot of the average spike per class from training data and the submission spikes predicted for each class

References

Draelos, R. (2019) "Best use of Train/Val/Test splits, with tips for Medical Data," *Glass Box* [Preprint]. Available at: https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/.

Mokri, Y. *et al.* (2017) "Sorting Overlapping Spike Waveforms from Electrode and Tetrode Recordings," *Frontiers in Neuroinformatics*, 11, p. 53. doi:10.3389/fninf.2017.00053.

Quiroga, R.Q. (2007) "Spike sorting," *Scholarpedia*, 2(12), p. 3583. doi:10.4249/scholarpedia.3583.

Rey, H.G., Pedreira, C. and Quian Quiroga, R. (2015) "Past, present and future of spike sorting techniques," *Brain Research Bulletin*, 119, pp. 106–117. doi:https://doi.org/10.1016/j.brainresbull.2015.04.007.

Shmueli, B. (2020) “Multi-class metrics made simple, part II: The F1-score,” *Multi-Class Metrics Made Simple, Part II: the F1-score* [Preprint]. Towards Data Science. Available at: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1.

Sunasra, M. (2019) “Performance metrics for classification problems in machine learning,” *Medium* [Preprint]. Medium. Available at: https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b.